13 August, 1999

An Automated Benchmarking Tool: The Analysis and Visualization Module

G. Marcais[1], A. Mink and M. Courson

Information technology Laboratory
National Institute of Standards and Technology (NIST)
Gaithersburg, MD 20899
amink@nist.gov

Abstract

Benchmarking and performance evaluation of high performance computing are a continuous on-going process that consumes significant stage time and generates large amounts of data to be stored and analyzed. The goal of this work is to propose an automated method to generate, capture and analyze an extensive performance profile. Although our initial efforts focus on commodity clusters, they are applicable to any parallel or distributed high performance computing system. Such an automated system has three main components: (1) a data collection and storage module, (2) a data analysis module, and (3) a run-time execution module. The focus of this report is on the data analysis module.

[1] Guest researcher from INT

TABLE OF CONTENTS

TABLE OF FIGURES

1. INTRODUCTION

Parallel computing has become commonplace in the high performance computing arena resulting in continual Quality of Service (QOS) evaluation for system tuning. Computing technology generations last only 18 months resulting in platform upgrade or replacement every 2 to 3 years. These factors require frequent running of benchmark codes which are used to evaluate new platforms or tune existing ones.

Different parallel machines are available at NIST for scientists to use: IBM SP2s, SGI Origin 2000s and Linux PC clusters [SALA99], [BEOW94]. In addition, our laboratory possesses a Linux PC cluster and a Windows NT cluster (under construction). MPI and PVM parallel environments are available on all these platforms.

Benchmarking these parallel architectures is more complex than for sequential machines as the number of factors and the amount of data involved increases. NIST is investigating the feasibility of developing an automated benchmarking toolset to reduce the manual effort involved in determining the various run configurations, instrumenting each program for performance data generation, running each configuration, collecting and storing the performance data and then retrieving the desired data for analysis and visualization. It is our intent to construct this toolset from a number of existing tools.

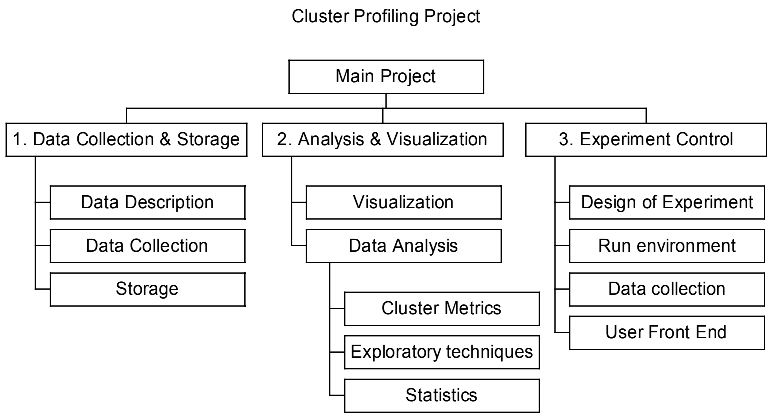


Figure 1 Project diagram.

The proposed benchmarking toolset consists of three modules as shown on Figure 1. The data collection and storage module [SIMO98] addresses the runtime collection and storage of performance data for later retrieval. The data storage organization is based on an abstract model of the cluster. Tools were reviewed to collect performance data and to store them efficiently.

This paper focuses on the analysis and visualization module. Figure 2 shows how this module integrates into the performance analysis process. Given an application and a hypothesis about this application, we design an experiment (i.e. a set of runs) to test this hypothesis. The ‘design of experiment’ box, or experiment builder, selects the runs based on the known dependencies between parameters and the existing runs in the database. The runs are then executed on the cluster and the performance data collected during execution stored in the database. The visualization and statistic module is then used to obtain insight about the application performance, which leads to the last stage of induction (i.e. determining what can be improved).

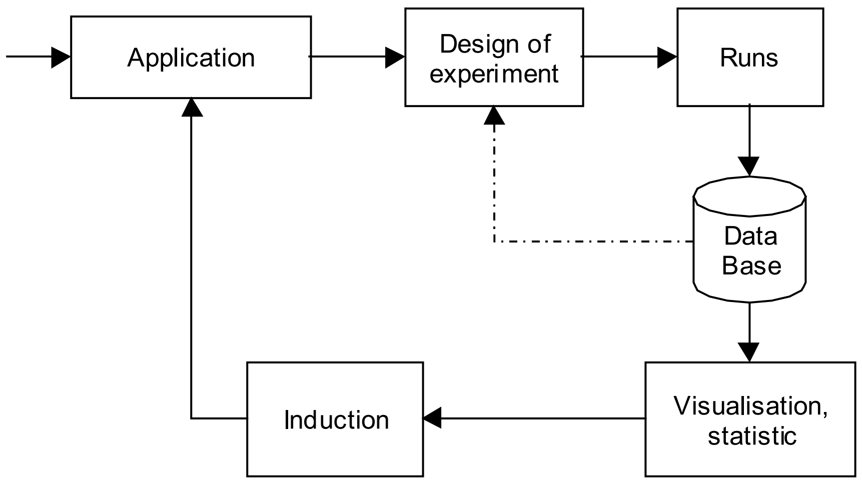


Figure 2 Performance study flow chart

2. Visualization tools

The features needed by our visualization module, are: the capability to plot any pair of raw data items from the database (e.g. memory usage against time), the capability to display the graphical statistical tests describe in section 3 and finally to plot the values from computed metrics (see section 4). The user interface will be a graphical interface.

The visualization tool can be a pure plotting program with no or few computation facilities and no statistical package. In that case, the computation work is handled by an encapsulating package that we would build. Alternately, it can be a complete data analysis environment with a powerful description language and embedded computation facilities. The user graphical interface (GUI) will be handle by the encapsulating package.

We chose the tools according to their features and their availability on different platforms (UNIX implementation, Win95/NT, MacOS). In addition, the feasibility of including them in an encapsulated package was taken in account (existence of an API). Finally, the capability to read data directly from a database (using ODBC or JDBC) was considered.

Table 1 summarizes the main features of the tools we investigated. The table lists the name of the tools, if they have a command line interface and a graphical interface (GUI), if they are commercially purchasable or a free product, if they have a database connectivity built-in, the name of the company or group who is developing the package and the list of supported platforms.

Tool	Cmd Line	GUI	Free[2]	DBMS	Vendor	Platforms
GnuPlot	Y	N	Y	N	Dartmounth college http://www.cs.dartmouth.edu/gnuplot_info.html.	UNIX, VMS, MSDOS
DataPlot	Y	Y	Y	N	NIST.[3] http://www.itl.nist.gov/div898/software/dataplot.html/language.htm.	UNIX, VMS, MSDOS, WIN95/NT
IDL	Y	Y	N	Y	Research Systems[4]. http://www.rsinc.com/	WIN95, MacOS, UNIX
MatLab	Y	Y	N	Y	MathWorks.[5] http://www.mathworks.com/	WIN95, MacOS, UNIX
Excel	N	Y	N	Y	Microsoft Corp[6]. http://www.microsoft.com.	WIN95, MacOS
SPlus	Y	Y	N	Y	MathSoft. http://www.mathsoft.com/	WIN95, UNIX

Table 1 Visualization tools with their main characteristics and platforms.

[2] All free software listed here are also open source code

[3] This software was developed by employees of the Federal Government in the course of their official duties. Pursuant to title 17 section 105 of the Unites States Code this software is not subject to copyright protection and is in the public domain

[4] IDL is a registered trademark of Research Systems, Inc.

[5] MatLab. Copyright 1984-1999 The MathWorks, Inc.

[6] Microsoft and Windows are either registered trademarks or trademarks of Microsoft Coporation in the U.S.A. and/or other countries.

2.1 *Pure visualization tools*

GNUPlot is a plotting tool. It does not include a statistical package or database connectivity. It can plot functions based on their equations or data from a file in 2D or 3D. It is configurable and relatively user friendly.

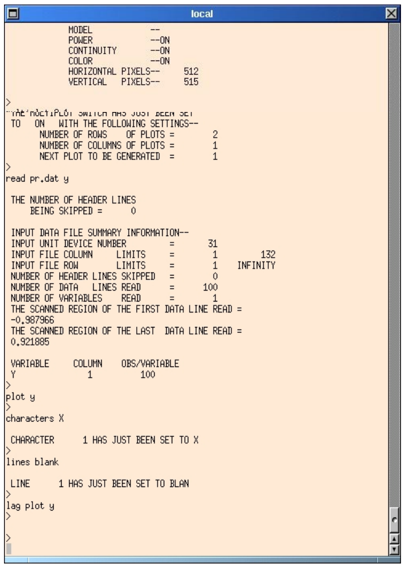
Although this tool does not contain any statistical capabilities or a graphical user interface (GUI), it is in this list because it is free and widely available to the UNIX community. It is small and the source code is available, which makes it one of the most popular tools for basic plotting. The ability to write programs and to drive GNUPlot from another application makes it easy to integrate in a larger tool as the plotting module.

2.2 *Data Analysis language*

DataPlot contains a very powerful language for data analysis and plotting. This language, based on English syntax, contains hundreds of statistical functions for running tests (e.g. t-test, F-test), computing statistical values (e.g. sample mean and confidence interval, etc.), using standard distributions (e.g. normal, exponential, etc.). This tool was designed by a statistician for statisticians, resulting in a command line interface and a graphical user interface front-end that is not very user-friendly. In addition, it does not have a built-in connectivity to a database.

DataPlot can be driven by another software, exactly the same way the GUI access the command line version of the software. Thanks to that, it might be easily integrated in a larger tool under UNIX. However, since EXPECT [EXPE96] is not supported on the WinNT/95 platform, integrating DataPlot is possible but it might require more work.

Figure 3 shows an example of loading data in DataPlot and then plotting the run sequence plot and the lag plot of this data (see 3.2).



```
            MODEL            --
            POWER            --ON
            CONTINUITY       --ON
            COLOR            --ON
            HORIZONTAL PIXELS--     512
            VERTICAL   PIXELS--     515

>
 THE MULTIPLOT SWITCH HAS JUST BEEN SET
 TO   ON   WITH THE FOLLOWING SETTINGS--
       NUMBER OF ROWS    OF PLOTS =        2
       NUMBER OF COLUMNS OF PLOTS =        1
       NEXT PLOT TO BE GENERATED  =        1
>
read pr.dat y

 THE NUMBER OF HEADER LINES
     BEING SKIPPED =      0

 INPUT DATA FILE SUMMARY INFORMATION--
 INPUT UNIT DEVICE NUMBER          =       31
 INPUT FILE COLUMN     LIMITS      =        1         132
 INPUT FILE ROW        LIMITS      =        1    INFINITY
 NUMBER OF HEADER LINES SKIPPED    =        0
 NUMBER OF DATA   LINES READ       =      100
 NUMBER OF VARIABLES    READ       =        1
 THE SCANNED REGION OF THE FIRST DATA LINE READ =
 -0.987966
 THE SCANNED REGION OF THE LAST  DATA LINE READ =
 0.921885

 VARIABLE     COLUMN    OBS/VARIABLE
 Y                1          100
>
plot y
>
characters X

 CHARACTER      1 HAS JUST BEEN SET TO X
>
lines blank

 LINE      1 HAS JUST BEEN SET TO BLAN
>
lag plot y
>

>
```

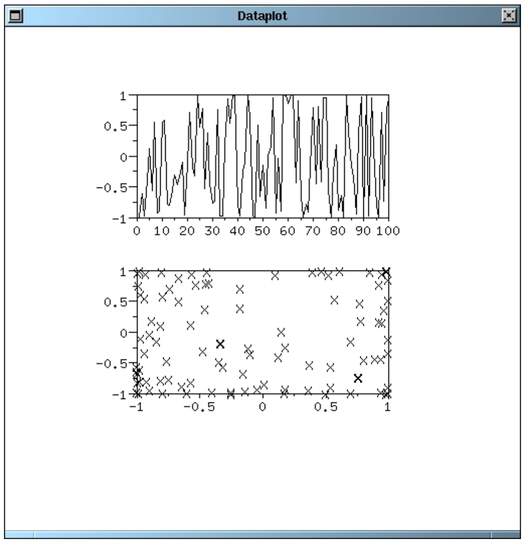


Figure 3 Run sequence plot and lag plot.

2.3 *General mathematical and drawing environment*

IDL and **MatLab** provide roughly the same approach. They are both based on an array/matrix oriented full-featured language. They can generate their own data or import from a file or a database, manipulate them and compute statistics. They can be run interactively or execute a program written in their own language or linked through their API to a program written in an external language (C, FORTRAN). Moreover, they both offer a GUI and tools to generate a program with a GUI.

They are not as statistically oriented tools as is DataPlot. However, they both have a statistical toolbox, powerful programs can be written with their own language and they are reasonably user friendly. They both offer a built-in connectivity to database (with ODBC and JDBC).

IDL's graphical interface is a graphical front-end as it stays mainly command line driven. Nevertheless, this graphical interface offers a built-in editor, a variable viewer and an enhanced command line interface. It is therefore a nice environment for both interactive use and developing IDL programs.

MatLab is a popular software package. It comes with plenty of add on packages (symulink, fuzzy tool box, etc.) which add a great number of tools to this environment. It is mainly command line driven (on all platforms) for standard operations. Many of the add on packages come with a built-in graphical interface, but all actions done with the graphical interface can be done with the command line and in Matlab's programs.

2.4 Spreadsheets

Spreadsheets programs offer a different view on data. They represent data on a large 2D table with relation between cells of this table. It is a convenient way of presenting data or generating new data. Most spreadsheets have visualization and statistical capabilities.

Excel is a spreadsheet with statistical and visualization features. It has only a graphical interface. It does not fit easily in a global package, as it does not offer a command line interface or stand-alone program. It is possible to interact with Excel using Microsoft's proprietary OLE technology and Visual Basic API but they poorly interact with the UNIX platform, our main development platform.

Nevertheless, it is the most widely used spreadsheet, offers easy point-and-click plotting capability and increasingly advanced statistical functions (regression, trend analysis, ANOVA).

Splus is a 2D-3D plotting program with statistics, data mining tools. It is based on the S object-programming language [SSYS98] from the Bell Labs specifically developed for data analysis. It has both a graphical and a command line interface for the WinNT/95 platform and only a command line interface for the UNIX platform.

Technically, Splus is not a spreadsheet, but it comes with a spreadsheet-like interface on the Windows graphical interface. Any action on the graphical interface has its equivalent in the S language.

We selected IDL and MatLab as they offer all the features required: 2D plotting, statistical features, database link, API. They are both able to plot raw data, perform graphical statistical analysis and to plot computed data. Therefore, we will use these existing computational features whereas the GUI will be mainly provided by the encapsulating tool.

3. Statistical analysis

This part proposes methods for the analysis of the collected data.

It focuses on graphical tests to see if our data meets the basic assumptions implied when using general statistics. Many different techniques exist to test the conformity of the data. We chose graphical techniques for their simplicity to conduct. We then evaluate techniques to make accurate systems comparison based on samples of data.

3.1 Measurement process assumptions

To acquire valid statistical analysis, we need to determine certain properties about our data, or assumed they are met. These properties assure that the experiment has been well conducted and therefore the statistics we obtain are meaningful. These properties are:

- *Fixed location.* The mean of the variable stays constant during the measurement process. This assures that the statistical mean is meaningful.
- *Fixed variation.* The range of the variable stays the same during the experiment. This is required to be able to compute the standard error.
- *Random drawing*. Each measurement is independent from all the other measurement. If not, the standard error is not valid.
- *Fixed distribution*. The random variable follows the same distribution while the measurement process. This is again to compute the standard deviation.
- *Normal distribution.* The sample is drawn from a population following a normal distribution.

If our data don’t meet these properties it may indicate an error during the experiment. We have to look more carefully at the experiment process: parameters evolution during the experiment, dependencies between different runs, number of runs. If some properties are still not meet, other more sophisticated statistics must be applied (not covered here).

3.2 Graphical tests

- The first two properties (fixed location and fixed variation) can be determined using a run sequence plot (the sample y versus its dummy index). The shape of the graph should be a stripe around a horizontal line with a constant thickness (Figure 4). A change in the location or the variation in the data indicates an evolution of the system under experiment.

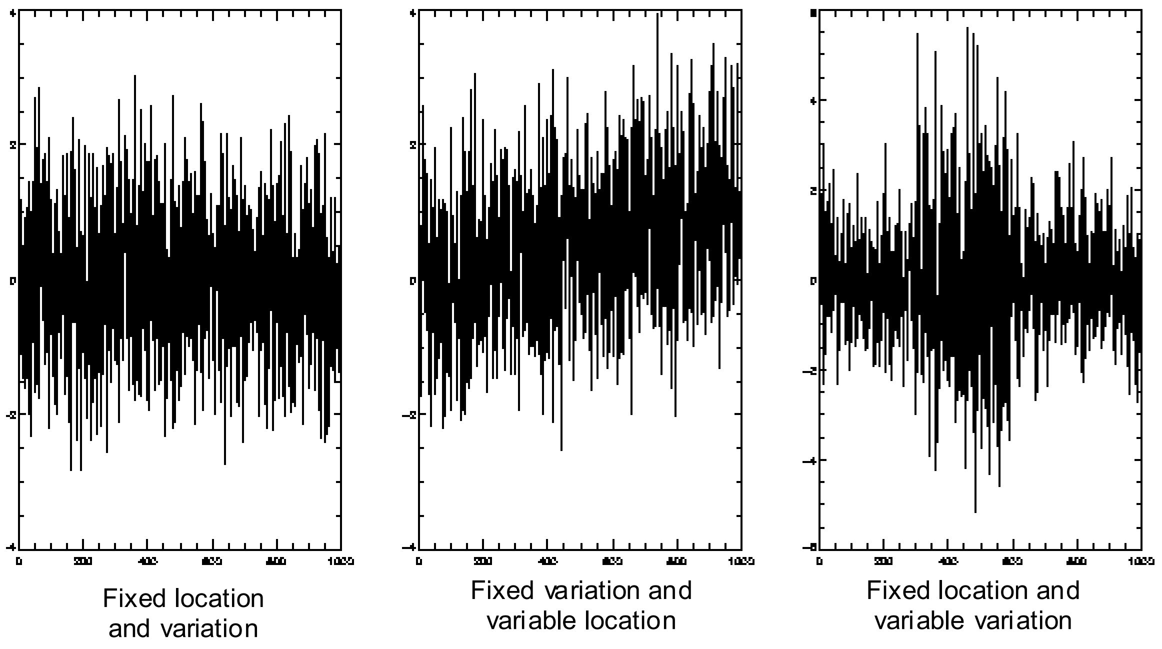


Figure 4 Different cases of run sequence plot to test fixed location and variation

- One can test the randomness of a set of sample using a lag plot. We plot the pairs $(y_i, y_{i+1}), \forall i \in [1, n-1]$, (n is the size of the set of samples). The shape of the graph should be like a cloud to indicate randomness. Ordered patterns on the lag plot indicate a correlation between samples (Figure 5). The independence between sample is required for most statistical test (see 3.3).

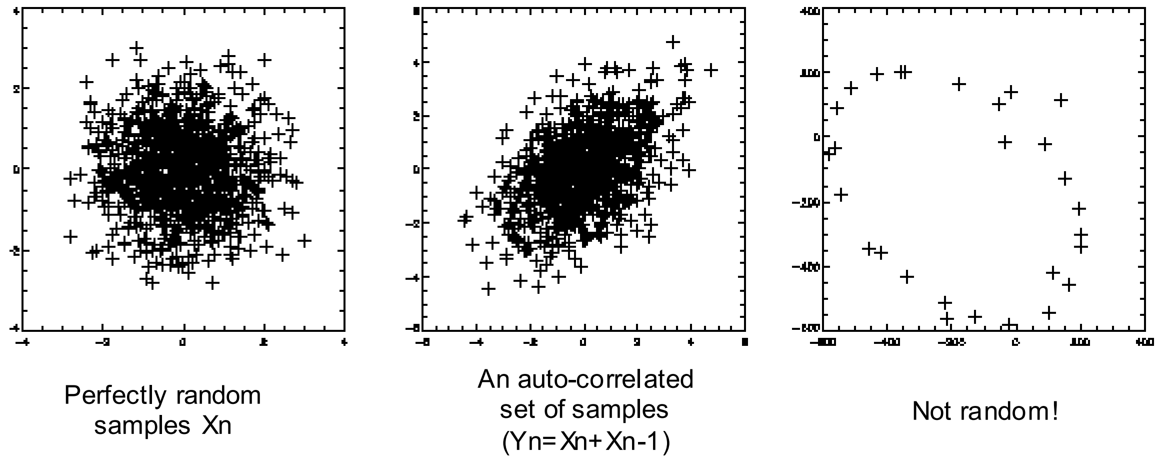


Figure 5 Different lag plot

- Plenty of different tests exist to test the normality of a set of samples (see [SHAP68], [FILL75]). The test described here consists in plotting in a quantile-quantile graph the observed quantiles versus the theoretical quantile. This method can be applied to any distribution. To be more precise lets consider F a cumulative distribution function and x_i the ith value out of n in our samples set. x_i is the i/n-quantile of our samples and we compute $y_i = F^{-1}(i/n)$ the i/n-quantile of the theoretical distribution. We then plot all the pair (x_i, y_i). The samples follow the theoretical distribution if the plot is linear (Figure 6). The normality is used to be able to conduct a t-test (see 3.3).

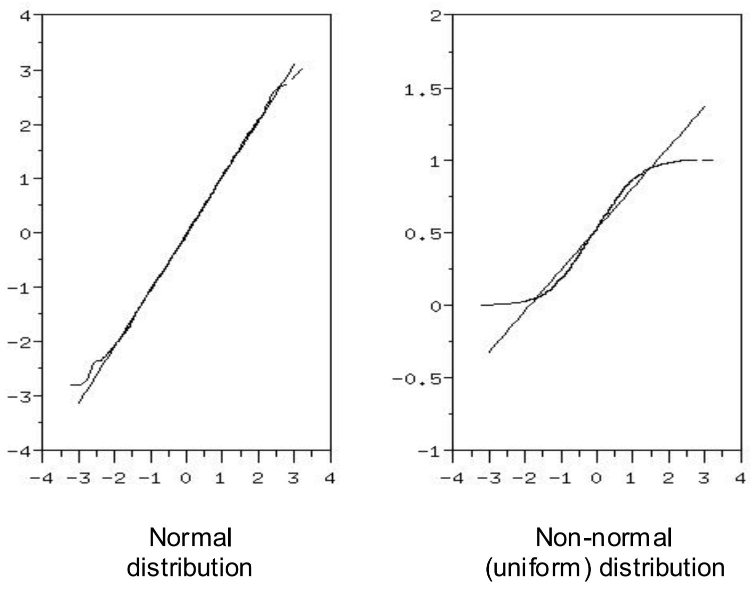


Figure 6 Normal probability graphs for two different distributions

3.3 Comparing systems using sample data

One major problem in statistic is we do not have access to the real value describing our data set (such as its mean, variance, distribution, etc.) but rather estimates of them. Moreover, each time we draw a sample from our population we obtain a different estimate. For example, we can compute the mean of a sample drawn from a population. This mean is different from the actual population mean but give us an estimate of it. Therefore, we never get one definitive value but rather a range of possible value in which the real value is likely to be. The size of this range is determined by the probability to be wrong in this estimation (i.e. the probability for the real value to be outside of the given range) chosen by the experimentalist.

In this section, we look at the techniques to compare two statistical results. In these tests, it is required that the samples are independent. One of the test also require that the sample are normally distributed. We can checked this properties with the tests of the last paragraph (see 3.2).

A (1-α)100% confidence interval for an estimated value is an interval (a, b) such as $P(a \le \mu \le b) = 1 - \alpha$, where μ is the estimated value. This means that the probability for the estimated value μ to be in the interval (a, b) is (1-α), and the probability to be out (i.e. to be wrong) is α.

The (1-α)100% confidence interval for the estimated mean of an independent sample of a large enough population (usually more than 30 elements) with an unknown variance is: $\left(\bar{x} - z_{1-\alpha/2}\frac{s}{\sqrt{n}}, \bar{x} + z_{1-\alpha/2}\frac{s}{\sqrt{n}}\right)$. Where:

- n the size of the sample,

- $\bar{x} = \frac{1}{n} \cdot \sum_i x_i$ is the estimation of the mean,
- $s = \frac{1}{n-1} \cdot \sum (x_i - \bar{x})^2$ is the sample variance of the samples,
- $z_{1-\alpha/2}$ is the (1-α)/2-quantile of a normal unit random variable.

Consequently, to compare a statistical value with zero, we have to look if zero is included in the (1-α)100% confidence interval or not. In the case of a positive answer, the statistical value can not be considered significantly different from zero at this level of confidence.

In the case we have less than 30 samples, we have to assume that the population is normally distributed to compute the confidence interval. Taking the same convention than before, the (1-α)100% confidence interval is: $\left(\bar{x} - t_{[1-\alpha/2;n-1]} s / \sqrt{n}, \bar{x} + t_{[1-\alpha/2;n-1]} s / \sqrt{n}\right)$. Where $t_{[1-\alpha/2;n-1]}$ is the (1-α)/2 quantile of the t-distribution with n-1 degree(s) of freedom.

To compare n pairs of observation, the same experiment must be repeated n times on each system. Then we compute the n differences of the elements in each pair, the mean of those differences and its confidence interval. Finally, we look if this mean is significantly different from zero (e.g. its confidence interval does not include zero). If it is not different from zero at the desired confidence level, the two systems performed quite in the same way. Otherwise, one system performs better (or worse) than the other.

The case of unpaired observations is more complicated and uses the t-test. Two set of samples, with the size n_1 and n_2 respectively, $\overline{x_1}$ and $\overline{x_2}$ are the mean for each set, s_1 and s_2 the variances. Then the (1-α)100% interval of confidence for the difference of the mean $\overline{x_1} - \overline{x_2}$, is:

$\left(\left(\overline{x_1} - \overline{x_2}\right) - t_{[1-\alpha/2;v]} \cdot s, \left(\overline{x_1} - \overline{x_2}\right) + t_{[1-\alpha/2;v]} \cdot s\right)$, where

$s = \sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}$ .is the variance and

$$v = \frac{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}{\frac{1}{n_1+1} \cdot \left(\frac{s_1^2}{n_1}\right)^2 + \frac{1}{n+1} \cdot \left(\frac{s_2^2}{n_2}\right)^2} - 2$$ is the number of degree of freedom.

Again, we look if the confidence interval includes zero or not to tell if the systems are significantly different.

The visualization tools selected in section 2 are able to conduct these tests and they will be use for that purpose in the analysis and visualization module.

4. Performance analysis

What the end user of the computer is concerned about is how long will it take to get the result. By parallelizing a program, we expect a smaller execution time for the same problem or a comparable execution time for a scaled problem size. For both cases, we are looking for a gain in speed with the parallel version compare to the sequential version.

Knowing the speed-up is not enough to diagnose an application in order to tune it. We need some other metrics that gives information about the internal behavior of the application. For example, metrics that link the communication and the computation part of the execution time.

We investigate the influence of many parameters on the speed up, like the overhead, the mean number of working processor etc. We give upper and lower bounds of the speed up given certain parameters and under what circumstances these bounds are reached (i.e. the best or the worst case).

We first describe of the parallel machine we have focused on. We then describe different parameters and their influence on the speed up. Finally, based on these relations, we discuss a type of diagram to analyze the performance of our parallel computer or program.

4.1 The parallel machine

The study conducted here is based on virtual parallel machines. It applies to clusters of interconnected computers (no assumptions are made about the network) on the Beowulf scheme [BEOW94]. Some of our assumptions may not be valid for shared memory systems.

We also take a high level point of view in this discussion as we only consider the computation time and the communication time. The communication time is the cumulative time spent in communication routines. It includes all the possible communication times between the nodes: system call, daemon (e.g. MPI daemon). We include the waiting time in the communication time since it is difficult to measure separately. The computation time includes anything that would be included in the sequential version (e.g. effective calculation, disk I/O, etc.).

So our cluster can be seen as a set interconnected nodes, each of them can be in either one of two states: computation state or communication state (see Figure 7).

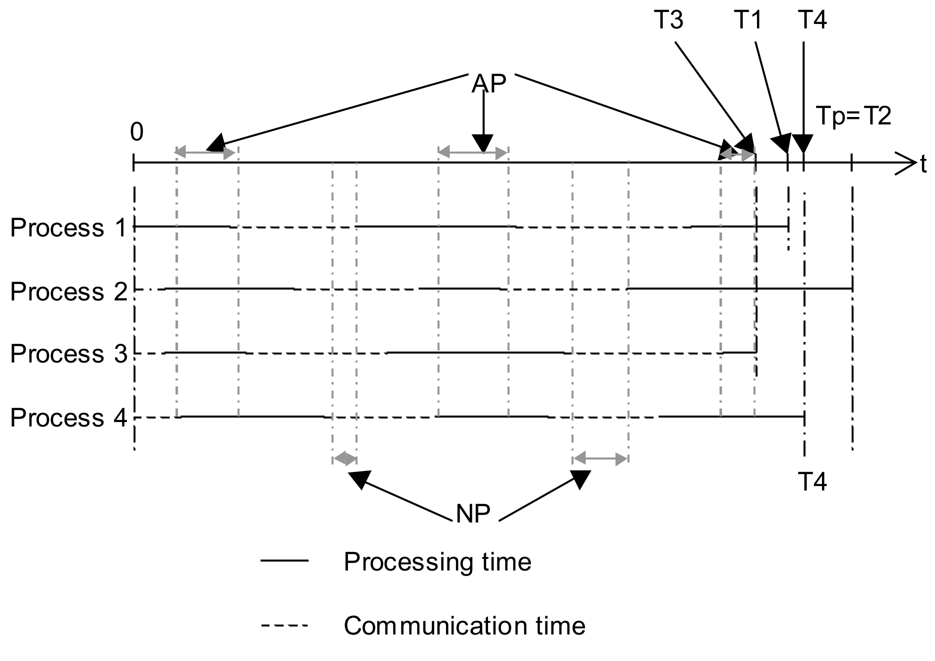


Figure 7 Application trace chart

4.2 Assumptions

Here are a few assumptions we have made for our parallel machine, and their direct consequences.

1) All the machines of the cluster are identical. This is not a significant constraint as heterogeneous clusters are rare for practical reasons. This case is not studied here.
2) We compare our parallel software performance with sequential software performance on the same machine. It is common to compare parallel performance with sequential performance.
3) The sequential program is pure computation, since there is no communication, and represents the minimum of work to do. This means we do not consider congestion problems or cache systems on computers. (See 4.3).

4.3 General metrics definitions

The response time is the time elapsed between the start of the program and its completion. It includes both computation and communication time.

The computation time of the sequential program is equal to its response time R_s. For each process in the parallel program, we take the notation (assuming we have n processors and $1 \le i \le n$):

X_i	communication time on the ith process
P_i	computation time on the ith process
$R_i = P_i + X_i$	response time of the ith process

The time X_i is considered as a global overhead time due to the parallel architecture and algorithm on processor i. We define the response time of the parallel program by the maximum of all the response times:

$$R_p = R_{\max} = \max_{1 \le i \le n}(R_i) .$$

Based on the assumption 1) that all the nodes are equivalent, there is a direct link between the computation time and the amount of work done. Two machines working independently on tasks for T units of time done the same work as a single machine working on them for 2T units of time. We speak indifferently here of time or amount of work.

We can now define two general metrics, similar to [DEEA89], the speed up and the efficiency:

$$SU = \frac{R_s}{R_p} \quad \text{the speed up}$$

$$EF = \frac{SU}{n} \quad \text{the efficiency}$$

$$\overline{R_p} = \frac{\sum_{i=1}^{n} R_i}{n} \quad \text{the average response time on the parallel program}$$

We use the notation from Ray Jan [JAIN91]. A HB (higher is better) metric is a metric whose high values are considered better than low values. A LB (lower is better) metric is a metric whose low values are considered better than high values.

The speed up and the efficiency are both considered as HB. It implies that we had an improvement in using a parallel version of the program, i.e. we get the result faster, if $SU \ge 1 \Leftrightarrow EF \ge 1/n$. The speed up represents our gain in using a parallel machine and the efficiency represents how efficiently the resources are used.

The definition of the average response time on the parallel program leads to: $\frac{R_p}{n} \le \overline{R_p} \le R_p$ (direct property of the mean). Moreover, the third assumption we made translates mathematically to the inequality:

$$\sum_{i=1}^{n} R_i \ge \sum_{i=1}^{n} P_i \ge P_0 = R_s$$

It tells that the amount of work done in the parallel program is at least what was done in the sequential one. We can obtain, with the last inequalities, a simple upper bound for the speed up and the efficiency:

$$R_s \le \sum_{i=1}^{n} R_i = n\overline{R_p} \le nR_p \Rightarrow SU \le n \,\&\, EF \le 1$$

We have the equality SU=n if $\overline{R_p} = R_p \,\&\, \sum_i R_i = R_s$. That means the load is very well balanced and there is no overhead due to the parallel program. In this case, the amount of work done in the parallel machine is exactly the same as the work done in the sequential one and every machine does exactly 1/nth of it with no parallel overhead. This case is rarely achieved and is referred to as “embarrassingly parallel”.

With our assumption, it is impossible to have $SU > n$, the so-called superlinear speed up. Other models of parallel machine show theoretically the possibility of the superlinear speedup ([HELM90], [GUST90]). We do not considerate this case here.

4.4 Other simple metrics

In this section, we propose different metrics which will help to provide an insight to the application’s behavior. We look at the links between these metrics and the speed up. This set of metrics will then be used in the performance discussion and the kiviat graphs (see 4.8)

Let AP and NP be defined as the time when all processor are working and no processor are working respectively (see Figure 7).

$$GOH = \frac{\sum_{i=1}^{n} R_i - R_s}{R_s} = \frac{n\overline{R_p} - R_s}{R_s} \quad \text{the global overhead}$$

$$POH = \frac{\sum_{i=1}^{n} P_i - R_s}{R_s} \quad \text{the computation overhead}$$

$$\eta = \frac{\sum_{i=1}^{n} X_i}{\sum_{i=1}^{n} P_i} \quad \text{communication - computation ratio (granularity)}$$

$$\overline{n} = \frac{1}{R_p} \sum_{i=1}^{n} P(i) \cdot i \quad \text{mean number of working processors}$$

The global overhead (GOH) is the relative difference in potential computation time: as $n\overline{R_p}$ ($n\overline{R_p} = \sum_{i=1}^{n} R_i$) represents the maximums amount of work the parallel machine could have done, the difference $n\overline{R_p} - R_s$ represents the overhead due to the parallelization of the program. This difference is normalized against R_s to obtain GOH. GOH is a positive value but has no upper bound. This is a LB metric.

The computation overhead (POH) represents how much time we spent in extra computation (due to different algorithms, different data representations, etc.) in the parallel version compared to the sequential program ($\sum_{i=1}^{n} P_i$ is the effective work done by the parallel machine). This value is again normalized against R_s and is a positive value but has no upper bound. This is a LB metric.

The η ratio represents the relative amount of time we spend in communications compared to the computations. It is a positive unbounded value. This is a LB metric.

In the last definition, $P(i)$ represents the amount of time when exactly i processors are in the computation state. The average number of working processors is a HB metric.

We have to be careful about the meaning of certain metric and their LB or HB characteristic. Albeit η is considered a LB metric, a low value by itself can be misleading. In fact, we can make η very low: if we give to all the processors the entire sequential job to do and we pick the first result to arrive (they should arrive roughly all together), the communication time is null, so is η. In this case the speed up is equal to one ($R_p = R_s$), that means no speed up, no gain in time but a very low η ratio.

4.5 Global overhead and speed up

The assumptions made in 4.2 yield to interesting properties for the metrics defined above. First of all, the global overhead (which can be easily measured) gives an upper bound and lower bound of the speed up (Figure 8):

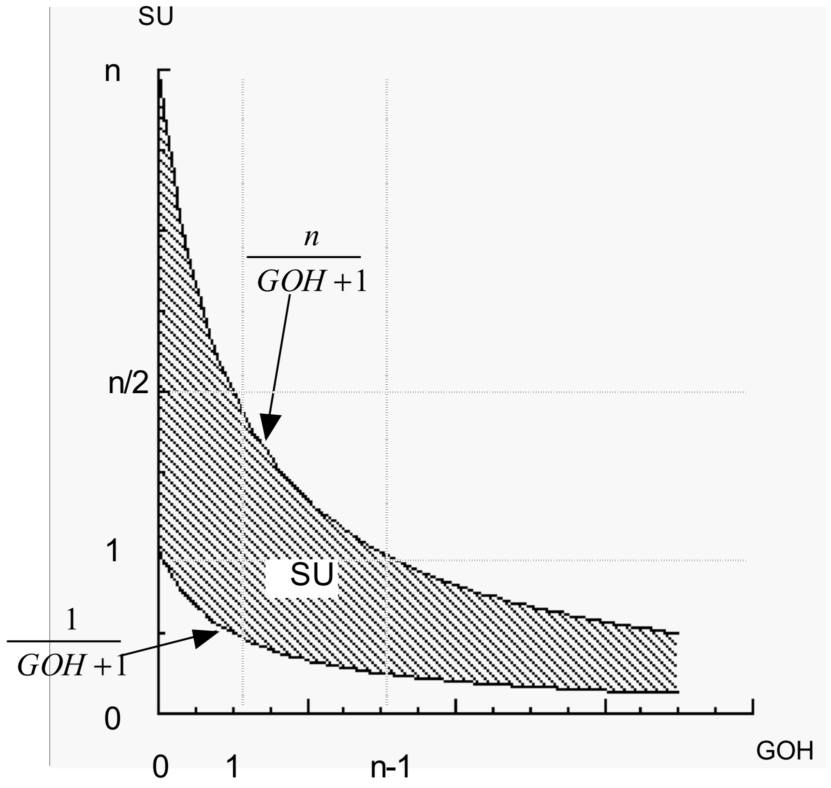


Figure 8 Relation between SU and GOH for n≥2 (graph made with n=4)

$$\frac{1}{SU}-1 \le \frac{R_p - R_s}{R_s} \le GOH = \frac{n\overline{R_p} - R_s}{Rs} \le \frac{nR_p - R_s}{Rs} = \frac{n}{SU} - 1 \Rightarrow$$
$$\frac{1}{GOH+1} \le SU \le \frac{n}{GOH+1}$$

The upper bound is reached if $\overline{R_p} = R_p \Rightarrow \forall i, j, R_i = R_j$. This is the perfectly well balanced case. The lower bound is reached if $n\overline{R_p} = R_p \Rightarrow \exists i_0, R_p = R_{i_0}\ and\ \forall i \neq i_0, R_i = 0$. This means that only one machine is working (machine i_0).

We can first say that $GOH \ge n-1 \Rightarrow SU \le 1$. If we expect any speed up the global overhead has to be less than n-1. Otherwise, it is useless to try to improve speed by using a parallel architecture. This condition is not very strong as $GOH \ge n-1 \Rightarrow \overline{R_p} \ge R_s$ which correspond to a very poor parallel algorithm (each node in the parallel version does, on average, more work than the sequential version).

Second, the load balancing on the cluster can be measured by the quantity $R_p - \overline{R_p}$. The higher this quantity is, the further from its upper bound the speed up will be.

4.6 Number of processor working and speed up

In this paragraph, we look at the influence of the last metrics on the speed up. First, we show that the average number of processors working is very simply linked to the speed up:

$$\bar{n} = \frac{1}{R_p} \sum_{i=1}^{n} P(i) \cdot i = \frac{1}{R_p} \sum_{i=1}^{n} P_i \Rightarrow$$

$$\bar{n} = \frac{R_s}{R_p} \cdot \frac{\sum_{i=1}^{n} P_i}{Rs} = SU \cdot \frac{\sum_{i=1}^{n} P_i}{Rs} = SU \cdot (POH + 1) \Rightarrow$$

$$\bar{n} \geq SU$$

We have equality $SU = \bar{n}$ if $\sum_{i=1}^{n} Pi = R_s \Leftrightarrow POH = 0$; e.g., the parallel algorithm implies no computation overhead. A necessary condition to obtain a speed up greater than one is to have $\bar{n} \geq 1$. Moreover, we have:

$$n - \bar{n} = n - \frac{1}{R_p} \cdot \sum_i P_i = \frac{\sum_i R_p - (R_i - X_i)}{R_p}$$

$$n - \bar{n} = \frac{\sum_i \left(R_p - R_i\right) + \sum_i X_i}{R_p}$$

So this difference represents the relative global amount of time the parallel program has spent in the wait state, including both the waiting time during the execution of the process ($\sum_i X_i$) and the waiting for the last process to end ($\sum_i (R_p - R_i)$).

We can therefore see $\bar{n}$ as an indication of how well the program is parallelized. It is a more effective and realistic upper bound of the speed up. It is the most speed up you can achieve with this algorithm due to its structure. This is the maximum computation time left, because the rest of the time is spent in waiting/communication. This maximum speed up can be reach if there is no overhead of computation. Otherwise, POH shows how much extra work this algorithm involves and how far the speed up is from its effective maximum.

We consider the influence of AP and NP on the speed up. These two measures are not as meaningful as the average number of processors computing. As we will see, the times when all the processors are working all together can be very small or even null and the speed up high anyway. On the other hand, NP can be null and the speed up very low. Nevertheless, they can be useful to diagnose a problem in the program in case the speed up is not as we expect it to be (e.g. dead locks in communication).

First, they are linked:

$$AP + NP \leq R_P \Rightarrow n \cdot \frac{AP}{R_p} \leq n\left(1 - \frac{NP}{R_p}\right)$$

For AP. The time when all the processors are working is included in every computation time (see Figure 7). So: $\forall i \; AP \leq P_i \Rightarrow n \cdot AP \leq \sum_i P_i$. With this first inequality, we obtain a lower bound of the speed up given the ratio of AP to the parallel response time R_p .

$$n \cdot \frac{AP}{R_p} \leq \frac{\sum_i P_i}{R_s} \cdot \frac{R_s}{R_p} \leq SU$$

For NP: the time when no processor is working is included in all communication time (symmetric case as AP). So: $\forall i, NP \le X_i = R_i - P_i \Rightarrow nNP \le \sum_i R_i - \sum_i P_i \le \sum_i R_i - R_s$. With this inequality we get:

$$n \cdot \frac{NP}{R_p} \le \frac{\sum_i R_i}{R_p} - SU \le n - SU \Rightarrow$$

$$SU \le n\left(1 - \frac{NP}{R_p}\right)$$

To have a speed up greater than one, the last condition requires that $\frac{NP}{R_p} \le 1 - \frac{1}{n}$.

Therefore, we have the inequality:

$$n \cdot \frac{AP}{R_p} \le SU \le n\left(1 - \frac{NP}{R_p}\right)$$

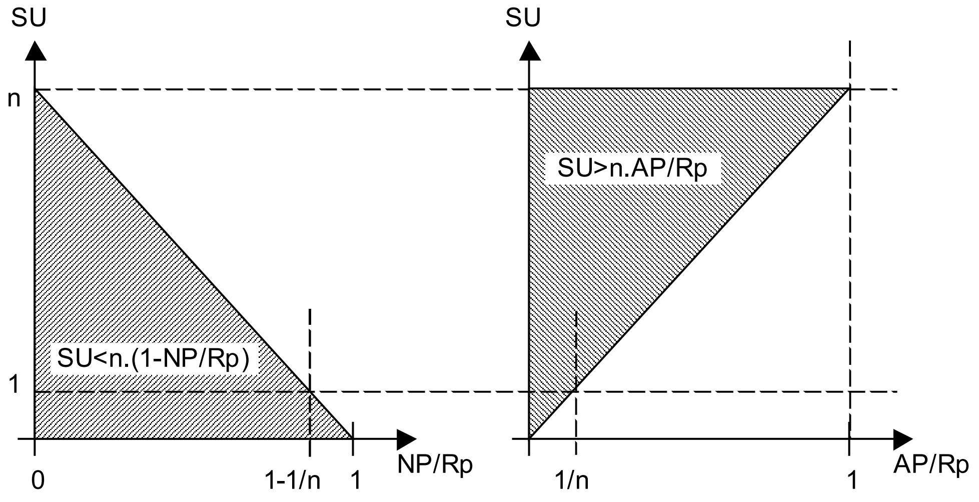


Figure 9 Relation between the speed up, AP and NP.

As we can see on the graphs of Figure 9, AP and NP give only rough indication on the speed up. NP can be very low and the speed up below one. Symmetrically, AP can be low and the speed very low. Only the right part of both graphs are meaningful.

4.7 The η ratio and the overhead

A relation between η and the overheads (global and computation). It is not directly linked to the speed up, but it is used to normalize η in the Kiviat graph (see 4.8).

$$\eta = \frac{\sum_i X_i}{\sum_i P_i} \ge \frac{\sum_i R_i - \sum_i P_i}{R_s} = \frac{\sum_i R_i - R_s}{R_s} - \frac{R_s - \sum_i P_i}{R_s} = GOH - POH$$, and

$$\sum_i X_i = \eta \cdot \sum_i P_i \Rightarrow \overline{R_p} = \frac{1}{n} \cdot \sum_i R_i = \frac{\eta+1}{n} \cdot \sum_i P_i \geq \frac{\eta+1}{n} \cdot R_s \Rightarrow$$
$$GOH \geq \frac{(\eta+1) \cdot R_s - R_s}{R_s} = \eta$$

Resulting in: $GOH \geq \eta \geq GOH - POH \geq 0$. In particular, this last inequality forbids having an important global overhead GOH, a small η ratio and a small computation overhead POH.

4.8 Tests and representations

We have shown several metrics and explained the kind of information they can provide about speed up. Based on the different parameters that bound the speed up, we provide a methodology to analyze our parallel software or architecture, shown Figure 10.

Error! Not a valid link.

Figure 10 Performance analysis flow chart

Kiviat graphs [MORI74] are a convenient way to present multiple metrics and detecting problems. The concept is not new but is applied here to parallel computing. The principle of these graphs is to put the metrics around a circle and alternate HB and LB metrics. There are typically six or height metrics (eight here) but graphs with more can be drawn. Therefore, an excellent system would yield a perfect star and a poor system a rotated star (see Figure 11). Other typical patterns give information about the application.

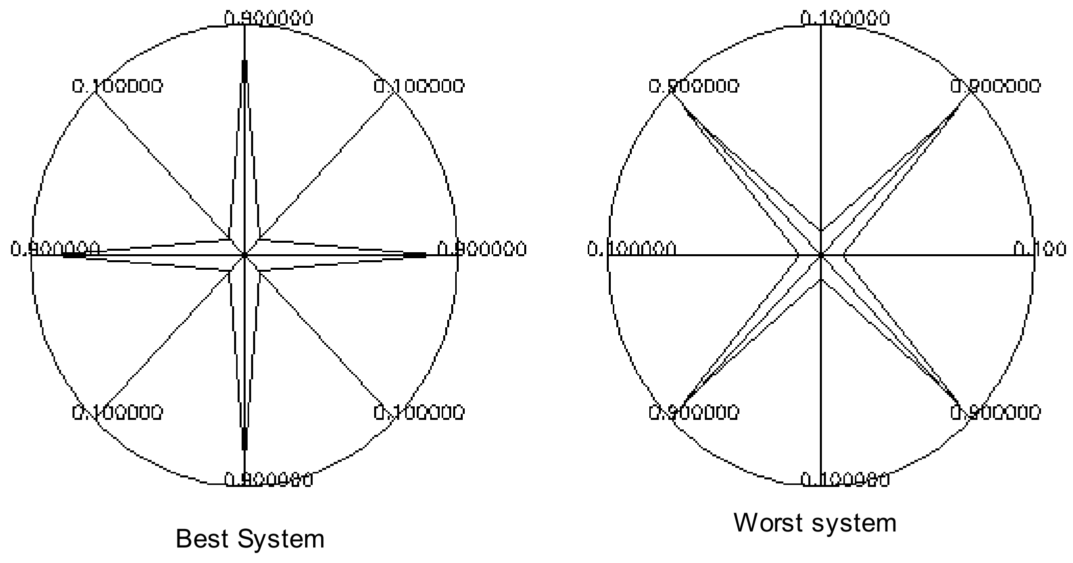


Figure 11 Kiviat representation of a perfect system and the worse system

We use the metrics from the previous sections, normalized to the range [0;1]. To normalize unbounded metrics, we assume that we are in the case $SU \geq 1$. In that case, the following inequalities are true: $GOH \leq n-1$, $POH \leq n-1$, $n - \overline{n} \leq n-1$ and $\eta \leq GOH \leq n-1$.

We use: $\frac{\overline{P}}{R_p}$, $\frac{\overline{X}}{R_p}$, $1-\frac{GOH}{n-1}$, $\frac{POH}{n-1}$, EF, $\frac{\eta}{n-1}$, $1-\frac{n-\overline{n}}{n-1}$ and $\frac{R_p-\overline{R_p}}{R_p}$; two metrics are artificially transformed from a LB metric to a HB metric by subtracting from one to complete the required symmetry of the Kiviat diagram (see Figure 12).

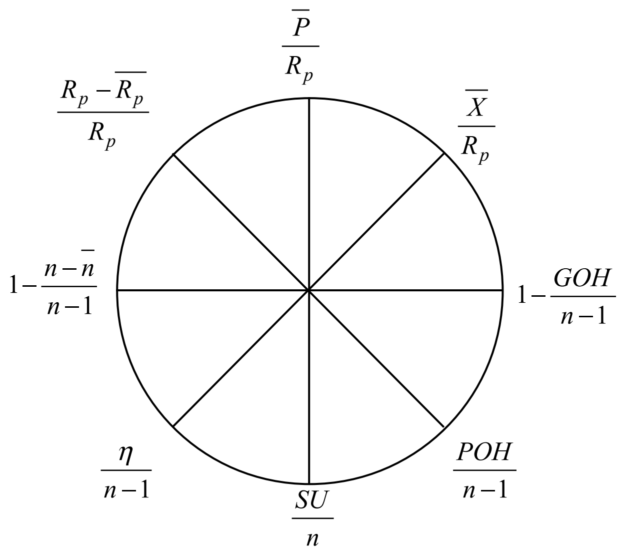

$\frac{\overline{P}}{R_p}$
$\frac{\overline{X}}{R_p}$
$1-\frac{GOH}{n-1}$
$\frac{POH}{n-1}$
$\frac{SU}{n}$
$\frac{\eta}{n-1}$
$1-\frac{n-\overline{n}}{n-1}$
$\frac{R_p-\overline{R_p}}{R_p}$

Figure 12 Description of the metrics around the Kiviat graph

5. Case Study

This NIST application, 3DMD, which we used for our study, solves a three dimensional Helmholtz equation [BENN93]. The program is composed of three computational phases each separated by a communication phase (i.e. two communication phases).

This program is written in C and uses the standard MPI communication environment [SALA98]. It was run four times on eight processor for each parallel environment: LAM from the Ohio Supercomputer Center[7] and MPICH from ANL/MSU[8]. The parallel machine is a cluster of 400MHz PentiumIIs running Linux interconnected by a Fast-Ethernet switch (100 Mb/s). The response time, communication and computation times for each process are collected using the MPIProf library[9].

Table 2 shows the measured response time of 3dmd on both parallel environments. We assume here that these values are coming from a normally distributed independent population but the size of the sample set is too small to test this assumption. We run a t-test on this data at the 95% confidence level to determine if one MPI implementation perform better than the other.

	LAM	**MPICH**	**DIF**
1	19.215345	18.884573	
2	19.948568	14.756383	
3	19.623048	17.062502	
4	19.31237	17.304259	
AVG	19.52483275	17.00192925	2.5229035
STDDEV	0.33172322	1.70116605	0.866603459
CONF	0.325082301	1.667109626	2.227671471

Table 2 3DMD on LAM and MPICH with 8 processors

As the confidence interval for the difference of the mean (2.52±2.22) does not contain zero, the two means are considered different at this level of confidence. We can then say here that the application performed better in the MPICH environment compared to the LAM environment.

Table 3 provides a list of our performance indicators (computation time and commutation time) as well as our proposed metrics. We plot each value of the corresponding indicator on a Kiviat graph (see Figure 13). This Kiviat graph shows that the MPICH version performed slightly better on all our metrics and our statistical t-test verifies that this result is statistically significant.

[7] http://www.mpi.nd.edu/lam/

[8] http://www-unix.mcs.anl.gov/mpi/mpich/

[9] http://cmr.nist.gov/mpiprof/index.html

LAM
nbProc 8

	Rp	avg(Ri)	sum(Pi)	sum(Xi)	SU	EF	GOH	POH	ETA	avg(n)	Rp-avg(Ri)	n-avg(n)
AVG	19.52483	16.17647	58.42359	70.9882	3.016997	0.377125	1.652214	-0.185474	1.215074	2.992936	3.34836	5.007064
STDEV	0.331723	0.39348	0.052321	3.165704	0.050967	0.006371	0.045061	0.355363	0.054569	0.05223	0.208699	0.05223
CONF	0.325082	0.385603	0.051274	3.102328	0.049946	0.006243	0.044159	0.348249	0.053476	0.051184	0.204521	0.051184

ALPHA 0.05

KIVIAT	avg(Pi)/Rp	avg(Xi)/Rp	1-GOH/(n-	POH/(n-1)	EF	eta/(n-1)	1-(avg(n)-r	(Rp-avg(Ri))/Rp
	0.374034	0.454474	0.763969	0	0.377125	0.173582	0.284705	0.171492

MPICH
nbProc 8

	Rp	avg(Ri)	sum(Pi)	sum(Xi)	SU	EF	GOH	POH	ETA	avg(n)	Rp-avg(Ri)	n-avg(n)
AVG	17.00193	14.93327	58.47791	60.98825	3.459407	0.432426	1.330724	-0.189741	1.042485	3.466074	2.068659	4.533926
STDEV	1.701166	1.845817	0.160293	14.626	0.360899	0.045112	0.233206	0.384846	0.247166	0.35602	0.969254	0.35602
CONF	1.66711	1.808865	0.157084	14.3332	0.353674	0.044209	0.228537	0.377142	0.242218	0.348892	0.94985	0.348892

ALPHA 0.05

KIVIAT	avg(Pi)/Rp	avg(Xi)/Rp	1-GOH/(n-	POH/(n-1)	EF	eta/(n-1)	1-(avg(n)-r	(Rp-avg(Ri))/Rp
	0.429936	0.448392	0.809897	0	0.432426	0.148926	0.352296	0.121672

Table 3 3DMD with LAM and MPICH performance results

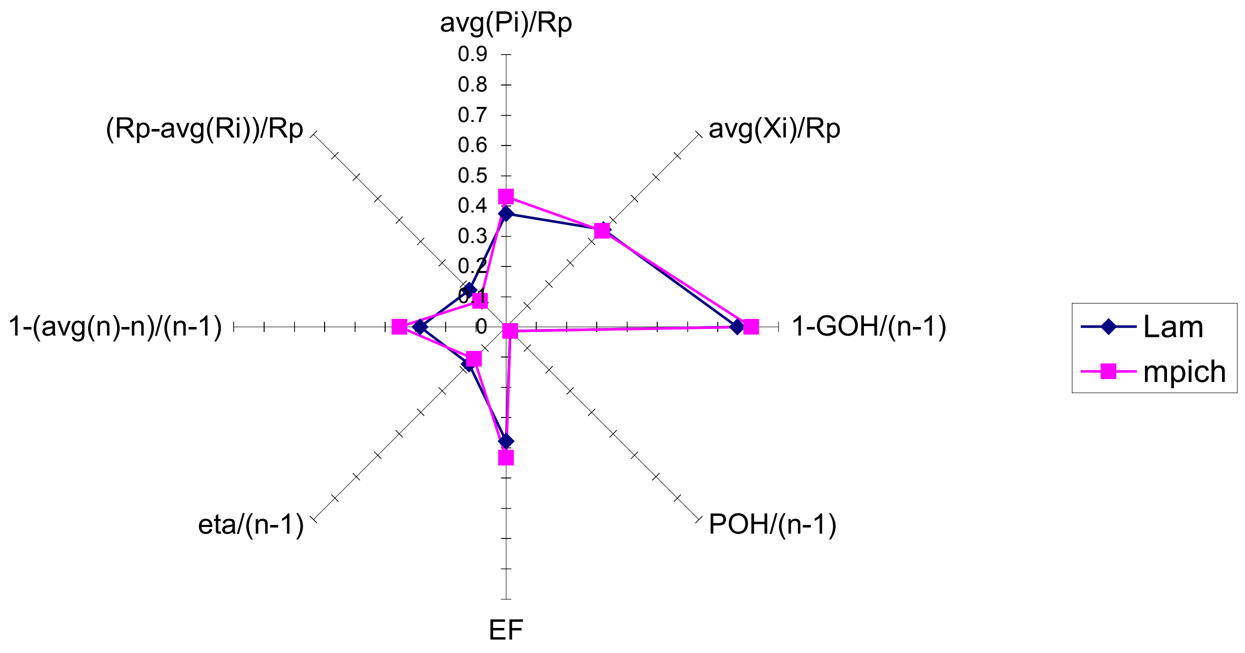


Figure 13 Kiviat graph for the LAM and MPICH environment. (NB: a value bigger than 0 zero plotted for POH/(n-1) to make it visible, but it is 0)

6. SUMMARY

We have evaluated different visualization tools and chosen two of them (IDL and MatLab) for our analysis and visualization module. We also presented a group of metrics to provide insight into the performance of parallel applications for evaluation and tuning.

We have a prototype implementation for our collection and storage module. We are now testing our metrics and the data collection tools on NIST applications and well-known benchmark programs. Finally, the experiment control, the third part of this project, is in progress.

For a future work, we are looking at methods to measure the waiting time. As noted in 4.1, it is difficult to distinguish waiting time from communication time, since waiting is distributed between the application, the MPI communication daemon and the operating system. We will investigate the feasibility of obtaining this information by comparing data from different sources.

Many metrics defined in this paper rely on the availability of the sequential response time (GOH, POH, SU). However, this value may not be available, for example if the sequential version takes too long to run. We are therefore looking at alternative ways to obtain insight into application and system performances.

BIBLIOGRAPHY

JAIN91 JAIN, R. (1991). The art of computer systems performance analysis, Wiley, New York.

PROB85 WALPOLE, R. E. & MYERS, R. H. (1985). *Probability and Statistics for Engineers and Scientists*, third edition, MacMillan, New York.

DEEA89 DEREK L., EAGER, JOHN ZAHORJAN & EDWARD D. LAZOWSKA (1989). Speedup versus efficiency in parallel systems, *Proceeding of IEEE,* volume 38, no 3, 408-423.

SIMO98 S. SIMON, M. COURSON (1998). *Cluster profiling project*.

BEOW94 CESDIS, "BEOWUL project at CESDIS". http://cesdis.gsfc.nasa.gov/

STAT78 G. E.P. BOX, W. G. HUNTER, J. S. HUNTER (1978). *Statistics for experimenters. An introduction to Design, Data Analysis, and Model Building.* John Willey & sons, New York.

EXPL99 J. J. FILLIBEN, (1999). *Exploratory Data Analysis,*.NIST conference.

EXPE96 D. LIBES, (1996). *Exploring Expect, a Tcl-base Toolkit for Automating Interactive Programs,* third edition, O'Reilly

SHAP68 S. S. SHAPIRO, M. B. WILK and H. J. CHEN (1968). *A comparative study of various tests for normality*. J. Am. Statist. Ass. 63, 1343-1372

FILL75 J. J. FILLIBEN, (1975). *The Probability Plot Correlation Coefficient Test for Normality.* Tehcnometrics, vol. 17, NO. 1, February 1975, 111-117

HELM90 D. P. HELMBOLD, C. E. McDOWELL (1990). *Modeling Speedup (n) Greater than n*. IEEE Transactions on parallel and distributed systems, vol. 1, NO 2, April 1990.

GUST90 J. L. GUSTAFSON (1990). *Fixed Time, Tiered Memory, and Superlinear Speedup*. IEEE Proceedings of the Fifth Distributed Memory Computing Conference, 1990. Volume: 2 , Page: 1255 –1260

BENN93 K. R. BENNETT, *Fast Direct Solution of Three-Dimensional Poisson and Helmholtz Problems on Distributed Memory Machines*, Proceedings of Sixth SIAM Conference on Parallel Processing for Scientific Computing, Norfolk, Viginia, March 22-24, 1993.

SALA98 W. SALAMON, A.MINK, M. INDOVINA, M. COURSON, *Evaluation of Applications on a Loosely-Coupled Cluster*, NISTIR 6148, April 1998.

NIST99 http://www.nist.gov/

SNEL97 R. SNELICK, M. INDOVINA, M. COURSON, A. KEARSLEY, "Tuning Parallel and Networked Programs with S-Check", *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'97), Las Vegas, Nevada* (June 30 - July 3, 1997), http://cmr.ncsl.nist.gov/scheck, 1997

MINK95 A. MINK, "Operating Principles of Multikron Virtual Counter Performance Instrumentation for MIMD Computers", *NISTIR 5743,* November 1995. http://www.multikron.nist.gov.

SALA99 W. J. SALAMON, A. MINK, "Linux Cluster at NIST", *Linux Journal*, pg 105-109, June 1999. http://www.linuxjournal.com

SSYS98 S system at Lucent Technologies. http://cm.bell-labs.com/cm/ms/departments/sia/S/index.html

MORI74 M. F. MORRIS (1974), Kiviat Graphs – Conventions and Figure of Merit, *Performance Evaluation Reviw*, **3**(3), 2-8.

ANNEXE

A Brief History of NIST [NIST99]

The National Institute of Standards and Technology (NIST), formerly the National Bureau of Standards (NBS), was established by Congress in 1901 to support industry, commerce, scientific institutions, and all branches of Government. For nearly 100 years the NIST/NBS laboratories have worked with industry and government to advance measurement science and develop standards.

NBS was created at a time of enormous industrial development in the United States to help support the steel manufacturing, railroads, telephone, and electric power, all industries that were technically sophisticated for their time but lacked adequate standards. In creating NBS, Congress sought to redress a long-standing need to provide standards of measurement for commerce and industry and support the "technology infrastructure" of the 20th Century.

In its first two decades, NBS won international recognition for its outstanding achievements in physical measurements, development of standards, and test methods -- a tradition that has continued ever since. This early work laid the foundation for advances and improvements in many scientific and technical fields of the time, such as standards for lighting and electric power usage; temperature measurement of molten metals; and materials corrosion studies, testing, and metallurgy.

Both World Wars found NBS deeply involved in mobilizing science to solve pressing weapons and war materials problems. After WWII, basic programs in nuclear and atomic physics, electronics, mathematics, computer research, and polymers as well as instrumentation, standards, and measurement research were instituted.

In the 1950s and 1960s, NBS research helped usher in the computer age and was employed in the space race after the stunning launch of Sputnik. The Bureau's technical expertise led to assignments in the social concerns of the Sixties: the environment, health and safety, among others. By the Seventies, energy conservation and fire research had also taken their place at NBS. The mid-to-late 1970s and 1980s found NBS returning with renewed vigor to its original mission focus in support of industry. In particular, increased emphasis was placed on addressing measurement problems in the emerging technologies. Many believe that the Stevenson-Wydler Act implemented, throughout the federal laboratories, the practices that had been developed at NBS over the years: cooperative research and technology transfer activities.

The Omnibus Trade and Competitiveness Act of 1988 -- in conjunction with 1987 legislation -- augmented the Institute's uniquely orchestrated customer-driven, laboratory-based research program aimed at enhancing the competitiveness of American industry by creating new program elements designed to help industry speed the commercialization of new technology. To reflect the agency's broader mission, the name was changed to the

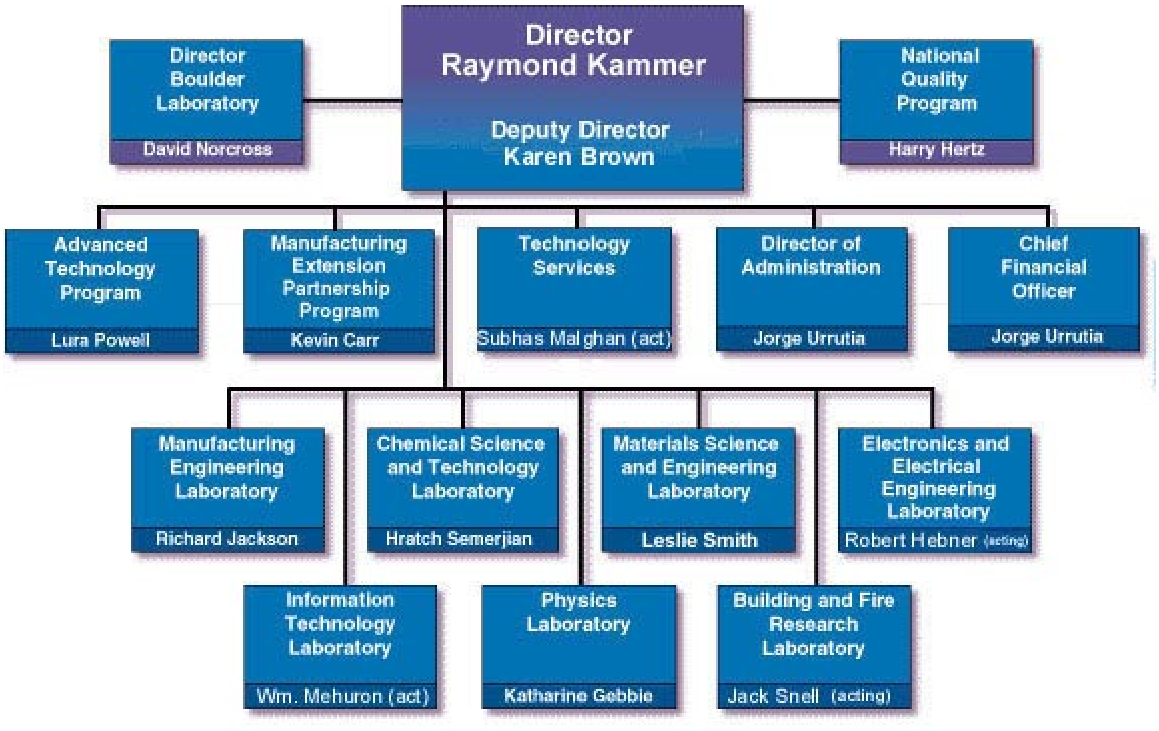
National Institute of Standards and Technology (NIST). These efforts, and the organizational changes brought by the NIST Authorization Act for 1989 which created the Department of Commerce's Technology Administration to which NIST was transferred, served as a critical examination of the role of NIST in economic growth. These mission and organizational changes, initiated under the Bush Administration were reaffirmed and strengthened by the Clinton Administration.

In addition to the reviews by Congress, the Administration, and the Department of Commerce, the Visiting Committee on Advanced Technology (VCAT) of NIST reviews and makes recommendations regarding the general policy, organization, budget, and programs of NIST. The VCAT holds four business meetings each year with NIST management, and summarizes its findings each year in an annual report that is submitted to the Secretary of Commerce and transmitted by the Secretary to Congress.

NIST's four major programs are designed to help U.S. companies achieve their own success, each one providing appropriate assistance or incentives to overcoming obstacles that can undermine industrial competitiveness. The programs are:

- Measurement and Standards Laboratories that provide technical leadership for vital components of the nation's technology infrastructure needed by U.S. industry to continually improve its products and services;
- a rigorously competitive Advanced Technology Program providing cost-shared awards to industry for development of high-risk, enabling technologies with broad economic potential;
- a grassroots Manufacturing Extension Partnership with a nationwide network of local centers offering technical and business assistance to smaller manufacturers; and
- a highly visible quality outreach program associated with the Malcolm Baldrige National Quality Award that recognizes continuous improvements in quality management by U.S. manufacturers and service companies.

NIST organization chart



Director
Raymond Kammer
Deputy Director
Karen Brown
Director Boulder Laboratory
David Norcross
National Quality Program
Harry Hertz
Advanced Technology Program
Lura Powell
Manufacturing Extension Partnership Program
Kevin Carr
Technology Services
Subhas Malghan (act)
Director of Administration
Jorge Urrutia
Chief Financial Officer
Jorge Urrutia
Manufacturing Engineering Laboratory
Richard Jackson
Chemical Science and Technology Laboratory
Hratch Semerjian
Materials Science and Engineering Laboratory
Leslie Smith
Electronics and Electrical Engineering Laboratory
Robert Hebner (acting)
Information Technology Laboratory
Wm. Mehuron (act)
Physics Laboratory
Katharine Gebbie
Building and Fire Research Laboratory
Jack Snell (acting)

Figure 14 Organization chart.

The Information Thechnology Laboratory divides itself in division and than in groups. I was working in the Scalable Parallel Systems & Applications in the High-Perfomance Systems & Services Division.

Information Technology Laboratory

An agency of the U.S. Department of Commerce's Technology Administration, the National Institute of Standards and Technology's primary mission is to promote U.S. economic growth by working with industry to develop and apply technology, measurements, and standards. The NIST's Information Technology Laboratory (ITL) is responding to the growing need for measurement and testing technology to support the development of computing and communications systems that are usable, scalable, interoperable, and secure. This need has come into sharper focus in recent years with the national effort to develop an information infrastructure and to support U.S. industry in a global information marketplace.

ITL has programs in three major areas:

- developing tests for human-machine interfaces, software diagnostics and performance, computer and network security, advanced network technologies, mathematical software, and conformance to standards;
- collaborating, consulting and operational services in computational sciences and information services; and 3.federal computer and network security activities;
- federal computer and network security activities.

High-Performance Systems & Services Division

The High Performance Systems and Services Division (895) of the Information Technology Laboratory enables effective application of high performance computing and communications systems in support of the U.S. information technology industry and NIST by: Conducting research, development and evaluation of advanced hardware and software components, new architectures, novel application technologies, and innovative measurement and test methods for improved computing performance, scalability, functionality, interoperability, flexibility, reliability and economy;

Serving as a testbed for R&D in high-performance computing and information technologies such as embedded computing, displays, and data storage, gaining experience in the deployment of these technologies, and developing metrics for the representative technologies;

Serving as a responsive, effective mission-critical resource spanning computational, communication, mass storage, security, archival, and scientific visualization services; and Providing and managing state-of-the-art computing and networking facilities which integrate and support an enterprise-wide heterogeneous information technology environment for NIST.